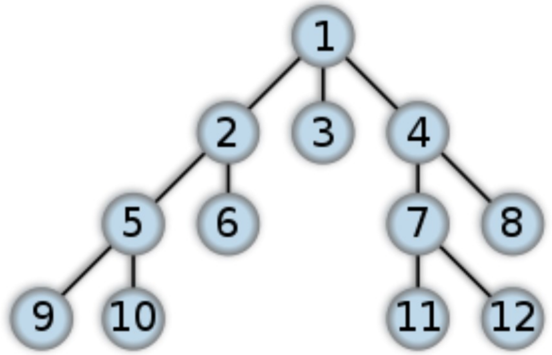
Breadth-first search

From Wikipedia, the free encyclopedia

Breadth-first search (**BFS**) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'[1]) and explores the neighbor nodes first, before moving to the next level neighbors.

BFS was invented in the late 1950s by E. F. Moore, who used it to find the shortest path out of a maze,[2] and discovered independently by C. Y. Lee as a wire routing algorithm (published 1961).[3][4]

Breadth-first search



Order in which the nodes are expanded

Class	Search algorithm
Data structure	Graph
Worst-case performance	$O(\lvert V\rvert + \lvert E\rvert) = O(b^d)$
Worst-case space complexity	$O(\lvert V\rvert) = O(b^d)$

Contents

Pseudocode

Input: A graph $Graph$ and a *starting vertex* $root$ of $Graph$

Output: Goal state. The *parent* links trace the shortest path back to $root$

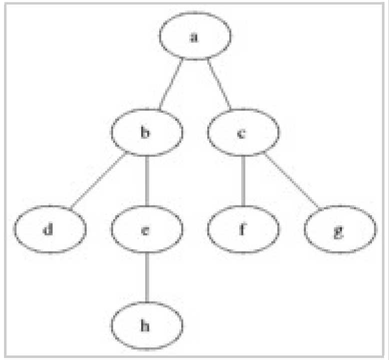
A non-recursive implementation of breadth-first search:

```
Breadth-First-Search(Graph, root):

    create empty set S
    create empty queue Q

    root.parent = NIL
    add root to S
    Q.enqueue(root)

    while Q is not empty:
        current = Q.dequeue()
        if current is the goal:
            return current
        for each node n that is adjacent to current:
            if n is not in S:
                add n to S
                n.parent = current
                Q.enqueue(n)
```



Animated example of a breadth-first search

More details

This non-recursive implementation is similar to the non-recursive implementation of depth-first search, but differs from it in two ways:

1. it uses a queue (First In First Out) instead of a stack and
2. it checks whether a vertex has been discovered before enqueueing the vertex rather than delaying this check until the vertex is dequeued from the queue.

The *Q* queue contains the frontier along which the algorithm is currently searching.

The *S* set is used to track which vertices have been visited (required for a general graph search, but not for a tree search). At the beginning of the algorithm, the set is empty. At the end of the algorithm, it contains all vertices with a distance from root less than the goal.

The *parent* attribute of each vertex is useful for accessing the nodes in a shortest path, for example by backtracking from the destination node up to the starting node, once the BFS has been run, and the predecessors nodes have been set.

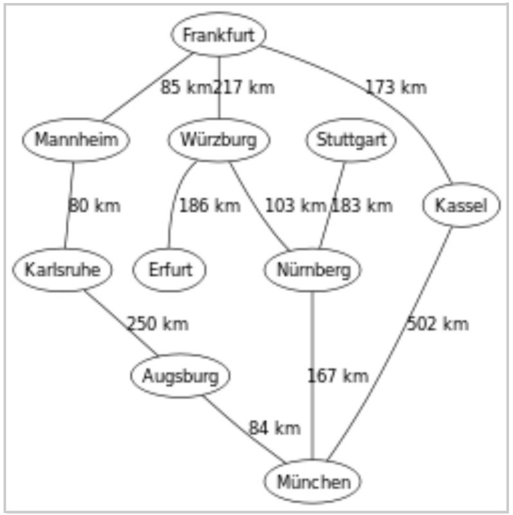
The *NIL* is just a symbol that represents the absence of something, in this case it represents the absence of a parent (or predecessor) node; sometimes instead of the word *NIL*, words such as *null*, *none* or *nothing* can also be used.

Note that the word *node* is usually interchangeable with the word *vertex*.

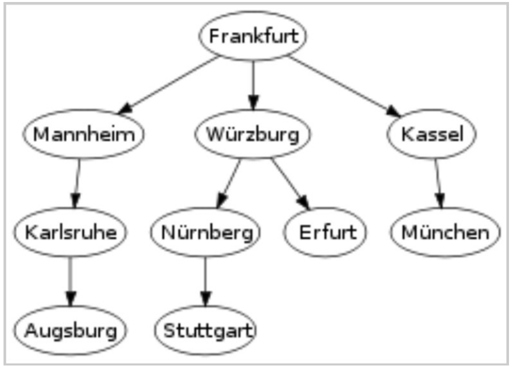
Breadth-first search produces a so-called *breadth first tree*. You can see how a *breadth first tree* looks in the following example.

Example

The following is an example of the breadth-first tree obtained by running a BFS starting from *Frankfurt*:



An example map of Germany with some connections between cities



The breadth-first tree obtained when running BFS on the given map and starting in Frankfurt

Analysis

Time and space complexity

The time complexity can be expressed as $O(|V|+|E|)$,[5] since every vertex and every edge will be explored in the worst case. $|V|$ is the number of vertices and $|E|$ is the number of edges in the graph. Note that $O(|E|)$ may vary between $O(1)$ and $O(|V|^2)$, depending on how sparse the input graph is.

When the number of vertices in the graph is known ahead of time, and additional data structures are used to determine which vertices have already been added to the queue, the space complexity can be expressed as $O(|V|)$, where $|V|$ is the cardinality of the set of vertices (as said before). If the graph is represented by an adjacency list it occupies $\Theta(|V|+|E|)$[6] space in memory, while an adjacency matrix representation occupies $\Theta(|V|^2)$.[7]

When working with graphs that are too large to store explicitly (or infinite), it is more practical to describe the complexity of breadth-first search in different terms: to find the nodes that are at distance d from the start node (measured in number of edge traversals), BFS takes $O(b^{d+1})$ time and memory, where b is the "branching factor" of the graph (the average out-degree).[8]:81

Completeness and optimality

In the analysis of algorithms, the input to breadth-first search is assumed to be a finite graph, represented explicitly as an adjacency list or similar representation. However, in the application of graph traversal methods in artificial intelligence the input may be an implicit representation of an infinite graph. In this context, a search method is described as being complete if it is guaranteed to find a goal state if one exists. Breadth-first search is complete, but depth-first search is not. When applied to infinite graphs represented implicitly, breadth-first search will eventually find the goal state, but depth-first search may get lost in parts of the graph that have no goal state and never return.[9]

BFS ordering

An enumeration of the vertices of a graph is said to be a BFS ordering if it is the possible output of the application of BFS to this graph.

Let $G=(V,E)$ be a graph with n vertices. Recall that $N(v)$ is the set of neighbors of v. For $\sigma=(v_1,\dots,v_m)$ be a list of distinct elements of V, for $v\in V\setminus\{v_1,\dots,v_m\}$, let $\nu_\sigma(v)$ be the least i such that v_i is a neighbor of v, if such a i exists, and be ∞ otherwise.

Let $\sigma=(v_1,\dots,v_n)$ be an enumeration of the vertices of V. The enumeration σ is said to be a BFS ordering (with source v_1) if, for all $1<i\leq n$, v_i is the vertex $w\in V\setminus\{v_1,\dots,v_i-1\}$ such that $\nu_{(v_1,\dots,v_{i-1})}(w)$ is minimal. Equivalently, σ is a BFS ordering if, for all $1\leq i<j<k\leq n$ with $v_i\in N(v_j)\setminus N(v_k)$, there exists a neighbor v_m of v_j such that $m<i$.

Applications

Breadth-first search can be used to solve many problems in graph theory, for example:

- Copying garbage collection, Cheney's algorithm
- Finding the shortest path between two nodes *u* and *v*, with path length measured by number of edges (an advantage over depth-first search)[10]
- (Reverse) Cuthill–McKee mesh numbering
- Ford–Fulkerson method for computing the maximum flow in a flow network
- Serialization/Deserialization of a binary tree vs serialization in sorted order, allows the tree to be re-constructed in an efficient manner.
- Construction of the *failure function* of the Aho-Corasick pattern matcher.
- Testing bipartiteness of a graph.

See also

- Depth-first search
- Iterative deepening depth-first search
- Level structure
- Lexicographic breadth-first search

References

1. "Graph500 benchmark specification (supercomputer performance evaluation)". Graph500.org, 2010.
2. Skiena, Steven (2008). *The Algorithm Design Manual*. Springer. p. 480. doi:10.1007/978-1-84800-070-4_4.
3. Leiserson, Charles E.; Schardl, Tao B. (2010). *A Work-Efficient Parallel Breadth-First Search Algorithm (or How to Cope with the Nondeterminism of Reducers)* (PDF). ACM Symp. on Parallelism in Algorithms and Architectures.
4. Lee, C. Y. (1961). "An Algorithm for Path Connections and Its Applications". *IRE Transactions on Electronic Computers*.
5. Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. p.597
6. Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. p.590
7. Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest. p.591
8. Russell, Stuart; Norvig, Peter (2003) [1995]. *Artificial Intelligence: A Modern Approach* (2nd ed.). Prentice Hall. ISBN 978-0137903955.
9. Coppin, B. (2004). Artificial intelligence illuminated. Jones & Bartlett Learning. pp. 79–80.
10. Aziz, Adnan; Prakash, Amit (2010). "4. Algorithms on Graphs". *Algorithms for Interviews*. p. 144. ISBN 1453792996.

- Knuth, Donald E. (1997), *The Art of Computer Programming Vol 1. 3rd ed.*, Boston: Addison-Wesley, ISBN 0-201-89683-4

External links

- Open Data Structures - Section 12.3.1 - Breadth-First Search (http://opendatastructures.org/versions/edition-0.1e/ods-java/12_3_Graph_Traversal.html#SECTION001531000000000000000)

Wikimedia Commons has media related to ***Breadth-first search***.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Breadth-first_search&oldid=769835791"

Categories: Graph algorithms | Search algorithms